CaVa^{DSL}: criação de Espaços Virtuais de Aprendizagem a partir de um repositório de documentos XML

Cristiana ARAÚJO

Universidade do Minho

decristianaaraujo@hotmail.com

Ricardo G. MARTINI

Universidade do Minho

rgm@algoritmi.uminho.pt

Pedro RANGEL HENRIQUES

Universidade do Minho

prh@di.uminho.pt

Maria João VARANDA PEREIRA

Instituto Politécnico de Bragança

mjoao@ipb.pt

Sumário

1. Introdução

"Instituições de Memória", como museus, arquivos ou bibliotecas, preservam hoje suas coleções, ou espólios, como Objetos Digitais (bases de dados ou documentos anotados). Após a digitalização e o registo, o objetivo imediato é explorar essas enormes fontes de informação relevantes que constituem a herança cultural da humanidade; isso requer pelo menos mecanismos de pesquisa precisos e poderosos mecanismos de publicação na Web. Museus virtuais —isto é, museus que não existem em uma localização física e não têm objetos físicos para mostrar —surgiram nesse contexto, da vontade de preservar a herança cultural juntamente com o potencial

aberto pela intranet. Em contrapartida os Museus Virtuais, exibem em suas salas de exposição objetos coletados de repositórios digitais.

Nesse caso, salas de exibição (que em nosso trabalho chamamos de Espaços de Aprendizagem - LS) são páginas da Web; o visitante acessa os objetos navegando em um *Web Browser* (Schweibenz, 2004).

Para criar um museu virtual na Web, é necessário consultar a implementaçãodigital do repositório e processar (transformar e relacionar) as informações retornadas antes de publicá-las como páginas da Web.

O trabalho relatado começa com uma discussão sobre comoimplementar ferramentas genéricas e eficientes capazes de extrair automaticamente os dados necessários (conceitos e relações) do repositório. De seguida, discutimos como construir as páginas da Web de um museu virtual de forma sistemática usando uma descrição formal de cada sala escrita em uma linguagem de domínio específico, $CaVa^{DSL}$, projetada para essa finalidade - —dessa forma, a plataforma de construção pode ser facilmente adaptada de um projeto para outro. $Cava^{gen}$ é o gerador que recebe as descrições formais do LS e cria as consultas para recuperar as informações do armazém de dados onde estão as instâncias da ontologia para exibi-las nas páginas Web, que realizam o desejado Espaço Virtual de Aprendizagem.

No projeto em discussão, interpretamos documentos anotados e construímos um filtro de texto capaz de criar automaticamente triplos que preencherão a ontologia do museu. Este filtro de texto converte documentos XML (eXtensibleMarkupLanguage) em notação RDF (ResourceDescription Framework).

Como um estudo de caso, para ilustrar a implementação deste processo e sua aplicação bem-sucedida, usaremos o repositório do Núcleo Português do Museu da Pessoa (Almeida et al., 2001; Simões

et al., 2003; Martini et al., 2016).

Na Secção 2, apresentamos a arquitetura CaVa proposta para atingir nosso objetivo: criar Espaços Virtuais de Aprendizagem (ou baseados na Web) a partir de um repositório digital. Após a visão geral do sistema, entramos em detalhes e, na Secção 3, discutimos o design e o desenvolvimento do filtro de texto, denominado tradutor XML2RDF, cuja função é transformar documentos XML em triplos RDF. A criação de Espaços Virtuais de Aprendizagem (VLS) e como extraímos as informações armazenadas na ontologia para exibir na Web (no VLS) é apresentada na Secção 4. Também nesta seção a abordagem é ilustrada apresentando algumas salas de exposição para o Museu da Pessoa, como um estudo de caso para testar os dois tradutores construídos. Finalmente, a Secção 5 apresenta a conclusão e aspossíveis linhas de trabalho futuro.

2. Arquitetura do sistema

O núcleo, ou coração, dessa abordagem é uma ontologia que modela o domínio do conhecimento relacionado com o museu a ser construído. A plataforma que será introduzida, o CaVa, pide o processo de construção em um primeiro módulo, a Função de Ingestão, para extrair dados das fontes e fazer upload dos triplos da Ontologia que descreve o universo cultural em torno do qual se quer construir o museuvirtual (Figura 1); e um segundo módulo, o Gerador ($CaVa^{gen}$), para gerar automaticamente a consulta para cada sala de exposição com base em uma especificação formal e um subconjunto da ontologia principal.

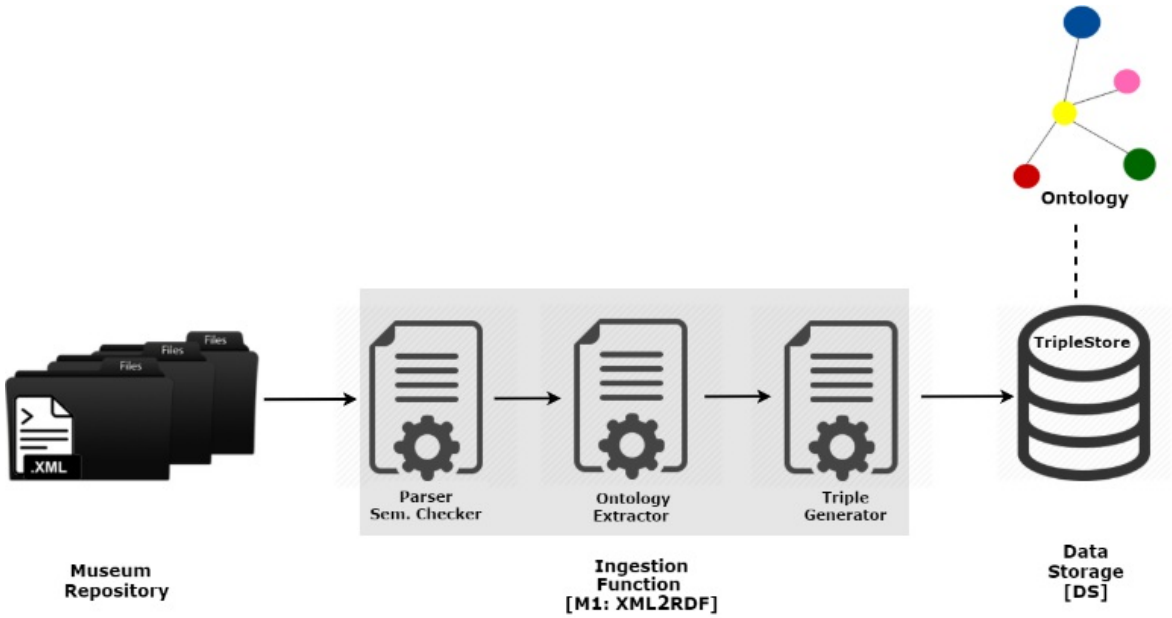

Figura 1. M1 – Extração de Dados e População da Base de Dados de Triplos

Além disso, o segundo módulo tem ainda de organizar as informações retornadas após o processamento da query emitida preparando os dados resultantes para serem exibidos em páginas da Web (Figura 2).

Como dito anteriormente, nossa abordagem pode ser caracterizada por uma arquitetura (representada nas Figuras 1 e 2) que compreende: o repositório; a Função de Ingestão [M1] responsável por ler os documentos anotados, extrair e preparar os dados e armazenar as informações coletadas; um sistema de armazenamento de dados (DS) que contém as instâncias da ontologia; uma Ontologia que descreve o domínio do conhecimento ligando os conceitos através de um conjunto de relações; o Gerador [M2] que recebe e interpreta os pedidos de informação, acedendo ao DS e retornando as respostas que são combinadas para configurar o VLS final (Araújo et al., 2016a; 2016b).

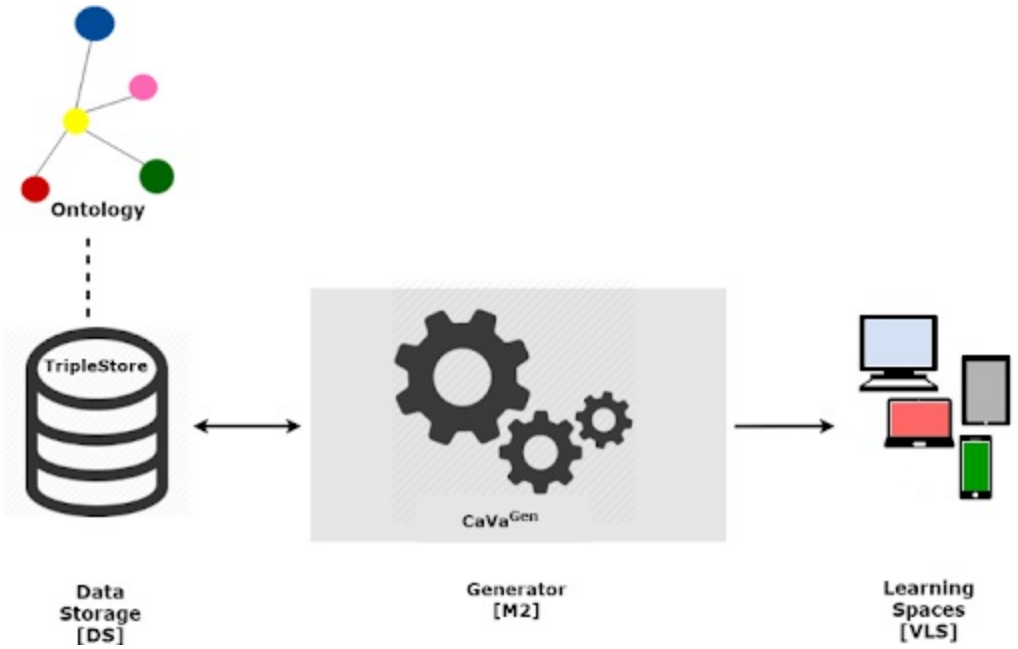

Figura 2. M2 - Geração de Ambientes Virtuais de Aprendizagem

3. Extração de dados e população da ontologia: XML2RDF

O papel do [M1] - TripleStorePopulation (Figura 1) é ler os documentos anotados, extrair e preparar os dados e armazenar as informações coletadas.

Assim, para desenvolver o M1, é necessário observar os elementos que podem aparecer nos documentos de entrada e sua estrutura, ou forma como se organizam, escrever um conjunto de regras de produção com base em expressões regulares e usar um gerador de filtros de texto para derivar o programa final.

Neste caso, a entrada é uma coleção estruturada de ficheiros XML (eXtensibleMarkupLanguage), e a saída será uma sequência de triplos (<sujeito, predicado e objeto>). Em cada triplo os conceitos (sujeito e objeto) correspondem aos valores dos atributos de um elemento XML ou até mesmo ao conteúdo do elemento. As relações

(predicado) que ligam os conceitos podem ser inferidas a partir dos elementos XML e sua estrutura (Araújo et al., 2017).

Como discutido acima, esse processo pode ser descrito usando um conjunto de regras de produção. Cada regra de produção é um par: no lado esquerdo, especificamos o elemento que queremos procurar - —expressão regular (RE); o lado direito é um código que transforma os dados de entrada (identificados e coletados pela RE à esquerda) e grava a respetiva saída (Araújo et al., 2017).

Para ilustrar a implementação desta proposta, usaremos os documentos pertencentes ao espólio do Museu da Pessoa (MP). O repositório digital do MP é composto por três tipos de documentos (BI - informações básicas, Legenda - –das respetivas fotos e Entrevista Editada).

A partir deste repositório, foi construída uma ontologia, utilizando os padrões CIDOC-CRM, FOAF e DBpedia, para armazenar as informações contidas no espólio, na forma de triplos. Para entender os conceitos que serão apresentados nos exemplos e imagens das próximas secções, a Figura 3 mostra um fragmento dessa ontologia. Não entraremos em mais detalhes sobre a ontologia, já uma vezque isso já foi discutido em outros artigos. Para mais informações sobre esta ontologia concreta, consulte: http://npmp.epl.di.uminho.pt/cidoc_foaf_db.html (Araújo et al., 2017).

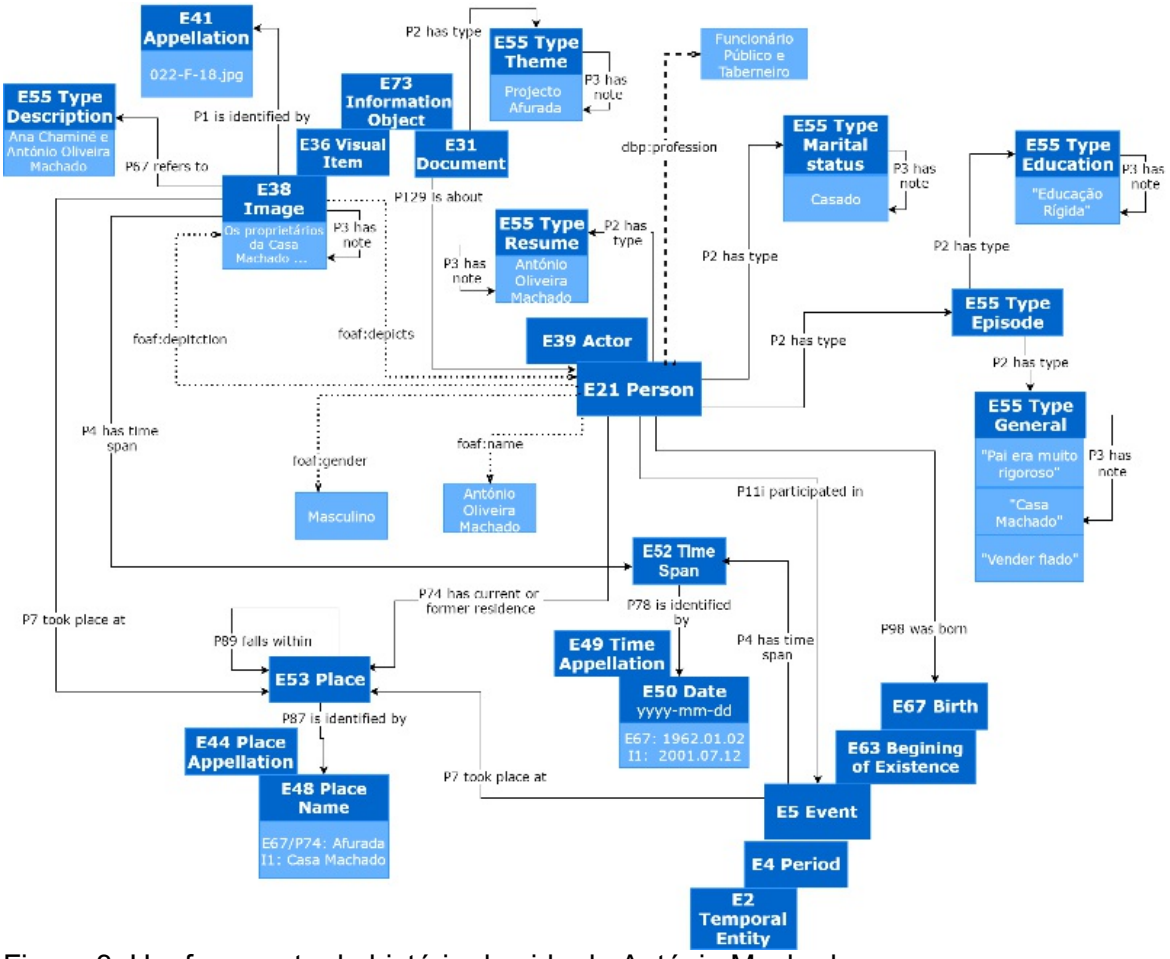

Figura 3. Um fragmento da história de vida de António Machado

Assim, para explorar o repositório digital mencionado acima, construímos um filtro de texto para processar os dados de entrada automaticamente, produzindo um repositório de triplos - XML2RDF (Araújo, 2016; Araújo et al., 2016b; 2018). Este filtro de texto foi desenvolvido usando ANTLR (Another Tool for LanguageRecognition). O ANTLR gera um analisador léxico que implementa o filtro de texto desejado para extração de dados, baseado em um conjunto de expressões regulares (Araújo et al., 2017).

Este filtro de texto recebe como entrada um documento XML. Após a análise e processamento, o tradutor emitirá uma descrição em RDF (Araújo et al., 2017).

A implementação do filtro de texto, XML2RDF, foi pidida em três

ficheiros, a saber: XML2RDF.g4, uma gramática (parte léxica) em ANTLR, organizada em '*modes*', que contém o conjunto de regras de produção (*RE-pattern / reaction*) que filtra o arquivos de entrada; Person.java, uma classe Java que define a representação interna para as informações que precisamos de extrair e processar sobre cada pessoa (um entrevistado); MainLexerXML2RDF.java o programa principal que orquestra os outros módulos para executar suas tarefas, a fim de implementar o tradutor (Araújo, 2016; Araújo et al., 2017).

A arquitetura do XML2RDF (o extrator de dados e o Gerador RDF, com base nesses três arquivos) é descrita na Figura 4. O ANTLR, a partir do ficheiro que contém a gramática XML2RDF.g4, gera a classe XML2RDF.java a qual será então compilada, incluindo a classePerson.java, para criar o processador XML2RDF desejado (Araújo, 2016; Araújo et al., 2017).

A Listagem 1 mostra a tradução automática, que é especificada por uma gramática ANTLR (Léxico). Esta figura mostra três regras de transformação (Cabec, Fotos e MP, que correspondem aos três arquivos de entrada acima referidos: BI, Legenda das Fotos e Entrevista Editada) para processar o início da especificação global (Araújo, 2016; Araújo et al., 2017; 2018).

```
lexer grammar XML2RDF;

Cabec   :  '<'[Bb][Ii]'>'              -> mode(sBI)
        ;
Fotos   : '<'[Ff][Oo][Tt][Oo][Ss]'>'   -> mode (sFOTOS)
        ;
MP      : '<'[Mm][Pp]'>'               -> mode (sMP)
        ;
Default:    .       { ; }
        ;
        ...
........Modes specification........
        ...
```

Listagem 1. Gramática (Léxico) XML2RDF em notação ANTLR

Cada regra tem um nome e um par que consiste em uma expressão regular e uma ação semântica escrita em Java. O papel da Expressão Regular é definir o padrão de texto a ser encontrado na entrada, e a ação semântica é especificar como o texto concreto será transformado (Araújo, 2016; Araújo et al., 2017; 2018).

Assim, quando o extrator lê uma tag XML que determina o início de um dos três ficheiros de entrada, ele entra em um modo ANTLR especial para processar o conteúdo desse documento (Araújo, 2016; Araújo et al., 2017; 2018).

Um exemplo de um modo ANTLR é mostrado na Listagem 2. Esse fragmento gramatical processa um episódio Geral, registado quando a pessoa entrevistada narra um episódio desse tipo.

```
1  mode sMP;
2  mode sEPISCab;
3  GetEPISCarac      :   [ ]*'caracter="'    -> mode(sEPISCarac)
4                     ;
5  GetEPISQuem       :   [ ]*'quem="'        -> mode(sEPISQuem)
6                     ;
7  GetEPISTitulo     :   [ ]*'titulo="'      -> mode(sEPISTitulo)
8                     ;
9  GetEPISTermo      :   [ ]*'termo="'       -> mode(sEPISTermo)
10                    ;
11 GetEPISTexto      :   '>'                 ->mode(sEPISTexto)
12                    ;
```

Listagem 2. Gramática (Léxico): Modo para lidar com 'Episódios' em uma entrevista

Neste caso, o extrator quando encontra a marca de abertura do bloco, que corresponde a um episódio dotipo geral, ativa o modo apropriado para processar o conteúdo deste bloco. Quando encontra a marca de fecho do bloco, o processador sai do modo específico e retorna ao modo inicial.

Os três modos auxiliares iniciais (linhas 3-10) contêm regras específicas para extrair informações dos atributos das etiquetas. O quarto modo auxiliar (linhas 11-12) contém regras específicas para

extrair a descrição do episódio de caráter geral.

Na Listagem 3, podemos ver as regras executadas (os modos ativados nas linhas 3-12 da Listagem 2) para analisar e extrair informações do repositório de documentos XML.

```
1  mode sEPISCarac;
2  GetCaracter     :   ~('"')+     {episCarac = getText();}
3                  ;
4  OutCaracter     :   '"'         -> mode(sEPISCab)
5                  ;
6
7  mode sEPISQuem;
8  GetQuem         :   ~('"')+     {episQuem = getText();}
9                  ;
10 OutQuem         :   '"'         -> mode(sEPISCab)
11                 ;
12
13 mode sEPISTitulo;
14 GetTITULO       :   ~('"')+     {episTitulo = getText();}
15                 ;
16 OutTITULO       :   '"'         -> mode(sEPISCab)
17                 ;
18
19 mode sEPISTermo;
20 GetTERMO        :   ~('"')+     {episTermo = getText();}
21                 ;
22 OutTERMO        :   '"'         -> mode(sEPISCab)
23                 ;
24
25 mode sEPISTexto;
26 GetEPISText     :   ~('<')+        {episTexto += getText();}
27                 ;
28 OutEPIS         :   '</episo'    -> mode(EPISSAVE)
29                 ;
```

Listagem 3. Gramática (Léxico): Modos Auxiliares

O fragmento de código entre as linhas 2 e 23 (Listagem 3) tem o papel de extrair informações dos atributos da tag XML. Mais especificamente, o tipo do episódio (linha 2-5), o narrador (linhas 7-11), o título do episódio (linhas 13-17), o termo do episódio (linhas 19-

23) são apresentados. A extração da descrição do evento é realizada nas linhas 25-29.

A geração do ficheiro de saída (RDF) é executada pelo fragmento gramatical mostrado na Listagem 4. O RDF que corresponde ao cabeçalho do episódio é impresso nas linhas 1-4. A descrição do episódio é impressa nas linhas 6-8. Nas linhas 10-15, o tipo do episódio é impresso, neste caso, é um episódio Geral.

```
1  if(general.size()>0){
2     System.out.println("<rdf:Description
      rdf:about=\"&ecrm;General_Interviewed_"+countinterview+"\">");
3     System.out.println("<rdf:type
      rdf:resource=\"&ecrm;E55_Type\"/>");
4     System.out.println("<P2_has_type
      rdf:resource=\"&ecrm;General\"/>");
5
6     for(String item: general){
7        System.out.println("<P3_has_note
      rdf:datatype=\"&xsd;string\">"+item+"</P3_has_note>");  }
8     System.out.println("</rdf:Description>");
9
10    if(!typeGeneral){
11       System.out.println("<rdf:Description
      rdf:about=\"&ecrm;General\">");
12       System.out.println("<rdf:type
      rdf:resource=\"&ecrm;E55_Type\"/>");
13       System.out.println("<P3_has_note
      rdf:datatype=\"&xsd;string\">General</P3_has_note>");
14       System.out.println("</rdf:Description>");
15       typeGeneral=true;
16    }
17 }
```

Listagem 4. Gramática (Léxico): Modo Print

Este fragmento gramatical é composto pelas regras executadas no final do processamento para imprimir os triplos RDF armazenados na representação interna.

Na próxima secção, detalhamos a geração automática de Espaços Virtuais de Aprendizagem (VLS) para exibir informações extraídas

pelo tradutor XML2RDF em um navegador da Web.

4. CaVa: Criação automática de ambientes virtuais de aprendizagem

Como dito anteriormente, CaVa é uma plataforma para gerar automaticamente Espaços Virtuais de Aprendizagem (VLS)com base em: uma ontologia, a qual descreve repositórios de informação institucional; e em uma especificação formal, escrita em $CaVa^{DSL}$ (Seção 4.1.), que define quais conceitos devem ser exibidos e como eles devem ser colocados nas páginas finais da Web.

As próximas secções tratam da especificação formal em$CaVa^{DSL}$ e do módulo principal do CaVa, chamado $CaVa^{gen}$, que é um conjunto de processadores que visa gerar o Ambiente de Aprendizagem virtual final.

4.1. $CaVa^{DSL}$: Especificando Ambientes de Aprendizagem

Uma Linguagem de Domínio Específico (DSL - Domain-Specific Language) é desenhada para ser facilmente usada quando é necessário descrever ou operar em um domínio particular ou em um conjunto de requisitos.

Neste caso, a linguagem $CaVa^{DSL}$ foi projetada para lidar explicitamente com a geração de Espaços de Aprendizagem na Web no âmbito do Patrimônio Cultural, envolvendo arquivos, museus e qualquer outra instituição cultural que deseja construir e organizar salas de exibição virtual com o intuito de expor para o público o conhecimento contido no repositório institucional.

CaVa^{DSL} foi concebida tendo em mente seu uso pelo curador de um museu, arquivista ou qualquer responsável da instituição cultural em questão. A sintaxe da linguagem é simples, mas expressiva, permitindo ao usuário final descrever as salas de exposição de maneira fácil.

CaVa^{DSL} é dividida em quatro blocos principais que descrevem a configuração da página principal, o cabeçalho, o conteúdo e o rodapé do LS.

- A configuração principal (*mainconfig*): especifica o título do LS e a descrição principal (por exemplo, o texto sobre a instituição cultural). Além disso, descreve outros componentes relacionados em geral com todo o museu virtual;
- Cabeçalho (*menu*): especifica o menu principal do LS. Ele compreende: o título que deve aparecer na barra do menu, as cores de segundo plano e de primeiro plano, o comportamento (se o menu deve ser fixo ou não) e o tipo de itens de menu (suspenso ou simples) com o rótulo e o link;
- O conteúdo (*exhibitions*): este componente da especificação descrevea lista de exposições que devem ser criadas. Cada exposição compreende: um título, uma breve descrição e um ícone; informação adicional com título e uma descrição; comportamento (se o componente da lista deve permanecer aberto (*expanded*) ou fechado (*collapsed*)); tipo de exposição (deve ser “permanente”, “temporário”, “futuro” ou “especial”); um operador de consulta (“all”: que procura no repositório todas as ocorrências do conceito especificado e retorna o conjunto de instâncias; “one”: que procura apenas uma instância (a primeira encontrada) que corresponda ao conceito indicado e que satisfaça o parâmetro condicional e o conceito da ontologia fornecido);
- O rodapé (*footer*): especifica uma área na parte inferior da página que compreende: imagens e data, nome da empresa ou do desenvolvedor, comportamento (como o componente de cabeçalho) e estilo (se o rodapé for simples com os dados

mencionados ou estendidos, e ainda se deve ter outras opções (por exemplo, link de redes sociais, etc.)).

Note-se que a CaVa^{DSL} pode ser estendida para incluir mais componentes, apenas é necessário criar novas produções na gramática (CaVa^{grammar}) que rege a linguagem. Por razões de espaço, a CaVa^{grammar} não será apresentada neste artigo.

Para exemplificar como descrever um elemento em CaVa^{DSL}, a Listagem 5 apresenta um fragmento de uma especificação para gerar o menu principal do Espaço de Aprendizagem “Museu da Pessoa”.

```
menu [
    brand: "Museum of the Person",
    background color: crimson,
    foreground color: white,
    behavior: fixed,
    options [
        label: "Exhibitions", dropdown [
            dropdown label: "All", url: "exhibitions",
            dropdown label: "Permanent", url: "permanent_exhibit",
            dropdown label: "Temporary", url: "temporary_exhibit",
            dropdown label: "Future", url: "future_exhibit",
            dropdown label: "Special", url: "special_exhibit",
        ]
        label: "About", url: "about", extension: php,
    ]
]
```

Listagem 5. Fragmento da especificação do menu em CaVa^{DSL}

A descrição apresentada na Listagem 5 especifica um Menu que possui um título, cores distintas no primeiro e segundo plano, um comportamento (fixo na parte superior da página). Além disso, é determinado que este menu terá doissubmenus: um submenudropdown chamado “Exhibitions”, que contém cinco sub-menus (“All”, “Permanent”, “Temporary”, “Future” e “Special”); e um submenu (simples) rotulado "About". Para cada um desses itens de menu, será criada uma página da Web correspondente. A Figura 5 apresenta o menu renderizado de acordo com a descrição da

Listagem 5.

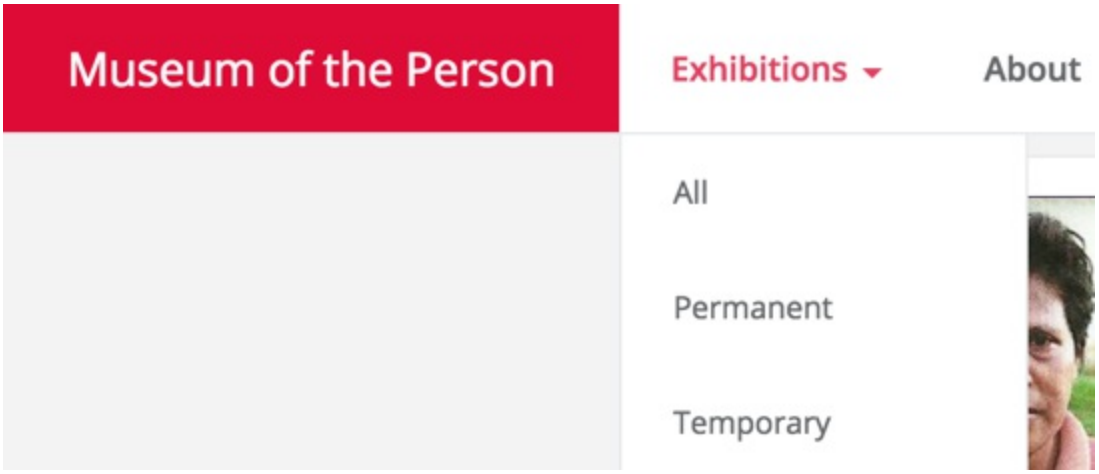

Figura 5. Menu principal do Espaço Virtual de Aprendizagem “Museum of the Person”

Para construir o menu principal desejado, mostrado na Figura 5, bem como o LS inteiro, é necessário um conjunto de processadores para analisar a especificação CaVa^{DSL} e produzir o código da página Web. Esses processadores têm o comportamento de um compilador, recebendo uma entrada e transformando-a em código da aplicação final (no nosso caso, *scripts* de páginas da Web (por exemplo, PHP, HTML, CSS, etc)). Na Seção 4.2. é explicado o papel do CaVa^{gen} para gerar um LS virtual de maneira sistemática, baseado em uma especificação CaVa^{DSL}.

4.2. CaVa^{gen}: Gerando Espaços Virtuais de Aprendizagem

CaVa^{Gen} é um conjunto de processadores que, com a entrada correta, produz os ficheiros de saída (conteúdo estático e dinâmico) que irão produzir o Espaço de Aprendizagem final, de acordo com a especificação CaVa^{DSL}. Nesta subsecçãodescrevem-se dois desses processadores: Processador CaVa, o núcleo do CaVa^{Gen}; e o

Processador CaVaSPARQLTriples, que lida com a geração de conteúdo dinâmico, ou seja, a geração de queries SPARQL.

O Processador CaVa transforma uma especificação $CaVa^{DSL}$ em vários arquivos (*scripts* Web) escritos em diferentes linguagens (HTML, PHP, JS, templates, CSS, etc.) que, quando colocados juntos, configuram várias páginas da Web, ou seja, o Espaço Virtual de Aprendizagem final. A figura 6 apresenta o esquema do processador CaVa.

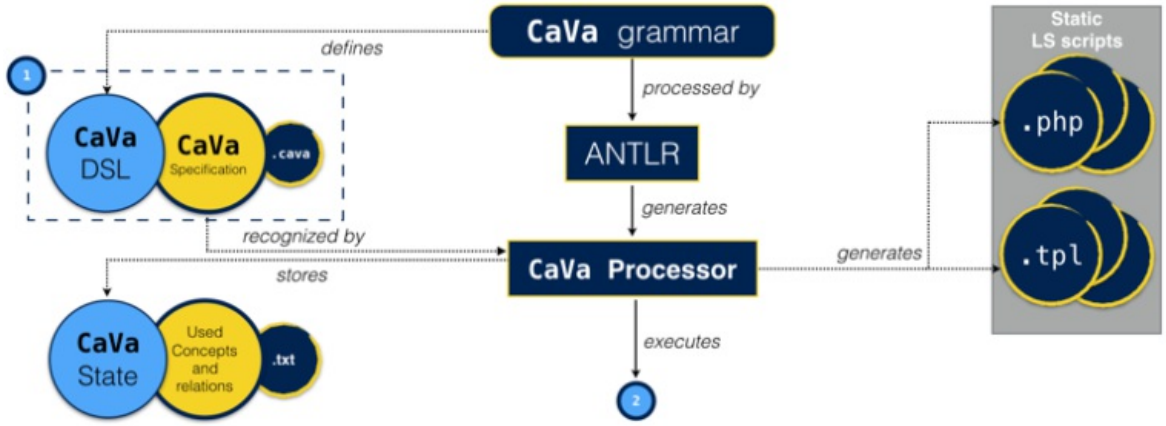

Figura 6. Esquema do Processador CaVa

O retângulo identificado pelo número (1) é o ficheiro de especificação (extensão ".cava") descrevendo um LS virtual baseado nas regras do $CaVa^{DSL}$. É a principal entrada para o processador CaVa.

A partir da especificação de entrada, o Processador CaVa gera o conteúdo estático do LS (basicamente os ficheiros .phpe.tpl (template)). Como pode ser visto na Figura 6, o Processador CaVaé criado pelo ANTLR, tomando como entrada a gramática $CaVa^{grammar}$. O círculo "CaVaState" representa os ficheiros de estado necessários para armazenar algumas configurações (no nosso caso, conceitos e relaçõesda ontologia que são referidos na especificação".cava") será criado um ficheiro de texto simples (.txt) a ser usado pelo Processador CaVaSPARQLTriples.

A implementação do Processador CaVa foi baseada nos *Listeners* do ANTLR (Parr, 2013). Basicamente, isso significa que, para cada produção de nossa gramática, existe um método *listener* que manipula o *token* reconhecido e produz alguma saída. O Processador CaVa normalmente recebe uma entrada, reconhece e gera código PHP (conteúdo estático). Por exemplo, recebendo a Listagem 5 como entrada, o Processador CaVa, por meio de um método *listener* chamado “enterHeader()”, produz o código mostrado na Listagem 6.

```
<?php
 $data = array(
  'brand' => "Museum of the Person",
  'bgColor' => "crimson",
  'fontColor' => "white",
  'behaviour' => "fixed",
  'options' => array(
     array('label' => "Exhibitions", 'dropdown' => "true",
        'dropdownListItems' => array(
           array('labelDropDown'=>"All",
   'urlDropDown'=>"exhibitions"),
           array('labelDropDown'=>"Permanent",
   'urlDropDown'=>"permanent_exhibit"),
           array('labelDropDown'=>"Temporary",
   'urlDropDown'=>"temporary_exhibit"),
           array('labelDropDown'=>"Future",
   'urlDropDown'=>"future_exhibit"),
           array('labelDropDown'=>"Special",
   'urlDropDown'=>"special_exhibit"),
        ),
     ),
     array('label'=>"About", 'url'=>"about", 'dropdown'=>"false"),
   ),
 );

 $tpl = new SMTemplate();
 $tpl->render('header', $data);
```

Listagem 6. Código PHP gerado baseado no código da Listagem 5

O código da Listagem 6 é armazenado em um ficheiro (neste caso, “header.php”) e, posteriormente, o conteúdo da variável *$data* é

passado para um ficheiro de template (chamado “header.tpl”), que contém alguns espaços reservados para lidar com o conteúdo de *$data* e renderizar o menu da Figura 5.

Note-se que a geração dos outros arquivos e conteúdo é semelhante ao processo de criação do menu do LS.

Além de gerador de conteúdo estático (Processador CaVa), o CaVa^{Gen} contém outro processador para gerar conteúdo dinâmico, ou seja, geração das consultas SPARQL e dos ficheiros das salas de exposições.

Como já mencionado, a CaVa^{DSL} permite, no bloco de conteúdo (exposições), a especificação de operadores de consulta. Quando um operador de consulta é definido em um componente de exibição, o Processador CaVa armazena as informações sobre a instrução e delega o processamento para o Processador CaVaSPARQLTriples, que é o processador que sabe como lidar com essas informações. Nesta fase, estamos preocupados com a geração de conteúdo dinâmico, mais precisamente queries SPARQL.

Para resolver a geração automática de queriesqueiresSPARQL, usamos uma abordagem que reutiliza gramáticas bem estabelecidas (RDF e Turtle). A Figura 7 apresenta o esquema do Processador CaVaSPARQLTriples que lida com essa tarefa.

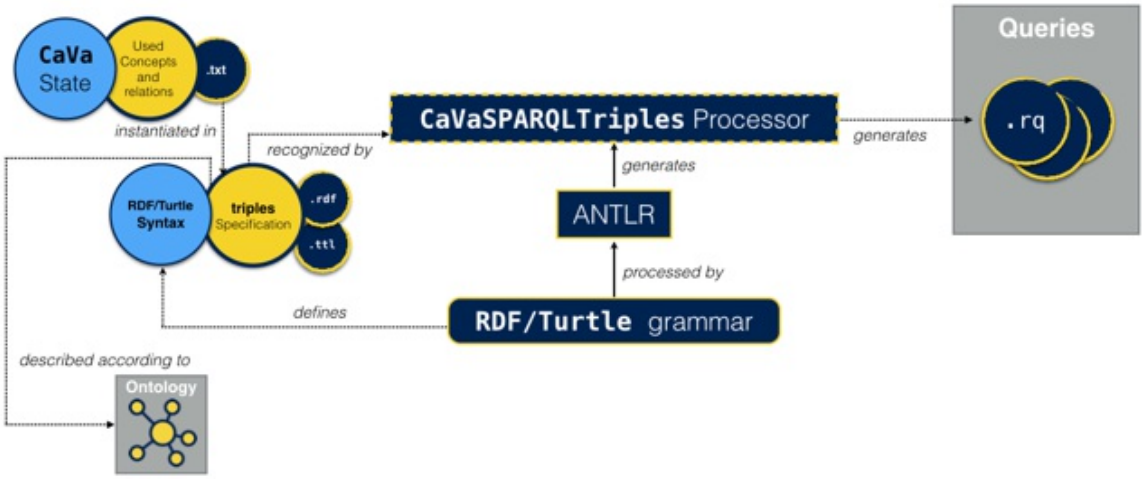

Figura 7. Esquema do Processador CaVaSPARQLTriples

A Figura 7 mostra o esquema completo para a geração de queries SPARQL (arquivos ".rq") com base nos conceitos e relações usadas (especificados no bloco "content" do $CaVa^{DSL}$) da ontologia.

Conforme apresentado na Secção 2., o $CaVa^{gen}$ recebe como entrada os triplos do Data Storage (DS) e, baseado na ontologia, reconhece os conceitos e relações relacionados ao operador especificado na especificação $CaVa^{DSL}$.

Assim, o Processador CaVaSPARQLTriples reconhece os triplos e cria a query SPARQL com base nesse conjunto de triplos que instanciam uma determinada ontologia. Para dar um exemplo do tipo de ficheiro de entrada do Processador CaVaSPARQLTriples, um fragmento RDF é mostrado na Listagem 7.

```
<rdf:Description rdf:about="General_Interviewed_11">
    <rdf:type rdf:resource="&ecrm;E55_Type"/>
    <ecrm:P2_has_type rdf:resource="General"/>
    <ecrm:P3_has_note rdf:datatype="&xsd;string">
      Nem todas as pessoas que frequentam a taberna Machado sao
      pescadores. Agora ja ha muita gente que vem de fora. Aos
      fins-de-semana, no Verao, nas ferias, ha pessoas que
      vem para a pesca desportiva, que vao pescar para a beira
      da Ponte da Arrabida e vem aqui comer qualquer coisa. Aos
      fins-de-semana temos aqui papas sarrabulho e vem gente de
      muito lados comer aqui. Mas vivemos mais a base das pessoas
      que vivem ca.
      A juventude ja foge das tabernas, nao para la.  Ja estao
      vocacionados para bebidas diferentes e nao se costumam
      embriagar. Nao quer dizer que nao haja um ou outro, mas
      sao casos isolados. A taberna vem gente de varias idades,
      vem de tudo, do mais novo ao mais idoso. Essa gente bebe
      cerveja e sumos, mas ainda ha ai alguns que  gostam mesmo
      de beber. Portanto e esse o modo de viver das pessoas.
    </ecrm:P3_has_note>
</rdf:Description>
```

Listagem 7. Um fragmento do arquivo de entrada RDF (triplos da ontologia

O código da Listagem 7 é apenas uma descrição RDF de uma única instância da ontologia usada (ou seja, “General_Interviewed_11”), que é uma instância da classe “E55_Type” e contém duas propriedades: uma propriedade de objeto “P2_has_type”, que está relacionada com o recurso “Geral”, que por sua vez é uma nova instância que também deve ser descrita no ficheiro RDF; e uma propriedade de tipo de dados “P3_has_note”, que está relacionada com os de tipos de dados e basicamente reflete um texto simples. Por causa do espaço, omitimos outras instâncias da ontologia neste artigo.

Com o objetivo de gerar o conteúdo dinâmico do Espaço de VirtualAprendizagem, o Processador CaVaSPARQLTriplesreconhece a entrada, como o fragmento na Listagem 7 e, com base nos *listeners* ANTLR, processa o código e produz uma saída que é uma query SPARQL. Assim, a partir do ficheiro de especificação $CaVa^{DSL}$, reconhecido e processado pelo Processador CaVa, quando um operador de consulta é encontrado no bloco de exibições, significa que o Processador CaVaSPARQLTriplespercebe que precisa de consultar o ficheiro de estado do CaVa (criado pelo Processador CaVa) para obter as informações necessárias para gerar a query SPARQL.

Tendo o conceito e as relações reconhecidas, o Processador CaVaSPARQLTriples executa uma sequência de ações para produzir a saída correta (query SPARQL) a ser utilizada no ficheiro da sala de exposições do EspaçoVirtual final. Essas ações são resumidas da seguinte forma:

1. Expandir cada instância do ficheiro RDF, selecionando apenas aquelas relacionadas com o conceito especificado na descrição $CaVa^{DSL}$;
2. Coletar e armazenar todas essas instâncias e informações importantes para a consulta final (expandidas na etapa 1);
3. Transformar cada nome de instância em uma nova variável da cláusula SPARQL WHERE (por exemplo, General_Interviewed_11, General, etc.). Também transformar

cada literal em uma nova variável da cláusula SPARQL SELECT, nomeando cada um deles, ?p__0, ?P__1, e assim por diante, dependendo de quantos literais são encontrados e importantes para a consulta;

4. Concatenar as duas cláusulas SPARQL (SELECT + WHERE) e os prefixos necessários (encontrados no ficheiro de entrada RDF) em uma *string*;
5. Criar e gravar o ficheiro com a query SPARQL (.rq) contendo o valor da *string* da etapa 4.

Após a execução dessas cinco etapas e com base no código da Listagem 7, a query SPARQL geradaé mostrada na Listagem 8.

```
PREFIX ecrm: <http://erlangen-crm.org/current/>

SELECT ?p___0
WHERE {
  ?General_Interviewed_11 a ecrm:E55_Type ;
  ecrm:P2_has_type ?General ;
  ecrm:P3_has_note ?p___0 .

  //specification of the ?General graph pattern
  . . .
}
```

Listagem 8. Consulta SPARQL gerada a partir do Processador CaVaSPARQLTriples

Depois de gerada a query SPARQL, ela precisa ser executada. Para executar essa ação, criamos outro processador, chamado *Query Processor*, que recebe o arquivo gerado ".rq" como entrada, pesquisa os resultados na base de dados de triplos e retorna o conjunto de resultados encontrados, armazenando-os em JSON para ser consumido pelo *script* da sala de exibição (no nosso caso, um ficheiro PHP gerado automaticamente pelo Processador CaVa) e passado para o navegador da Web para renderizá-lo com base em um template (.tpl). Na Figura 8 é apresentada a sala de exposição final gerada, mostrando um episódio narrado pela instância "General_Interviewed_11".

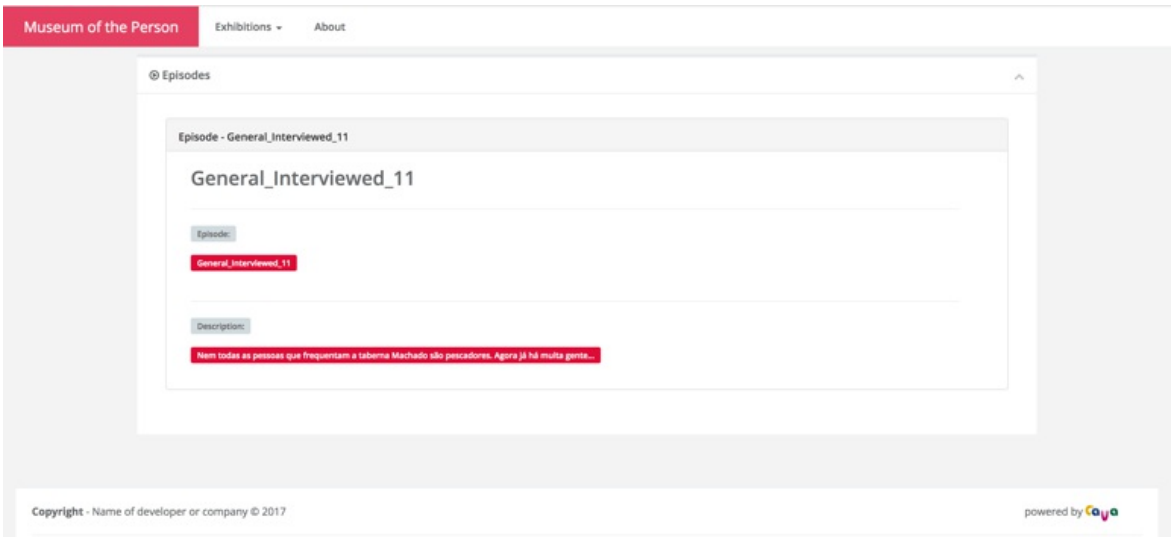

Figura 8. Sala de Exibição final do Museu da Pessoa

Como pode ser visto na Figura 8, há um menu superior (cabeçalho) semelhante ao já mostrado no exemplo do Processador CaVa, um rodapé e o conteúdo da sala de exposição no meio do ecrã, exibindo os resultados obtidos pelo Processador de Queries (*Query Processor*) e processado pelo *script* PHP da sala de exposição, preenchendo os espaços reservados do template com os resultados da query SPARQL.

Isso configura uma sala de exibição completa dentro de um Espaço Virtual de Aprendizagem, onde o usuário pode navegar sobre as instâncias e os conceitos da ontologia de acordo com a exposição concebida pelo curador.

Para concluir, esta abordagem foi usada para implementar todas as salas de exibições do Museu da Pessoa cuja página principal é apresentada na Figura 9.

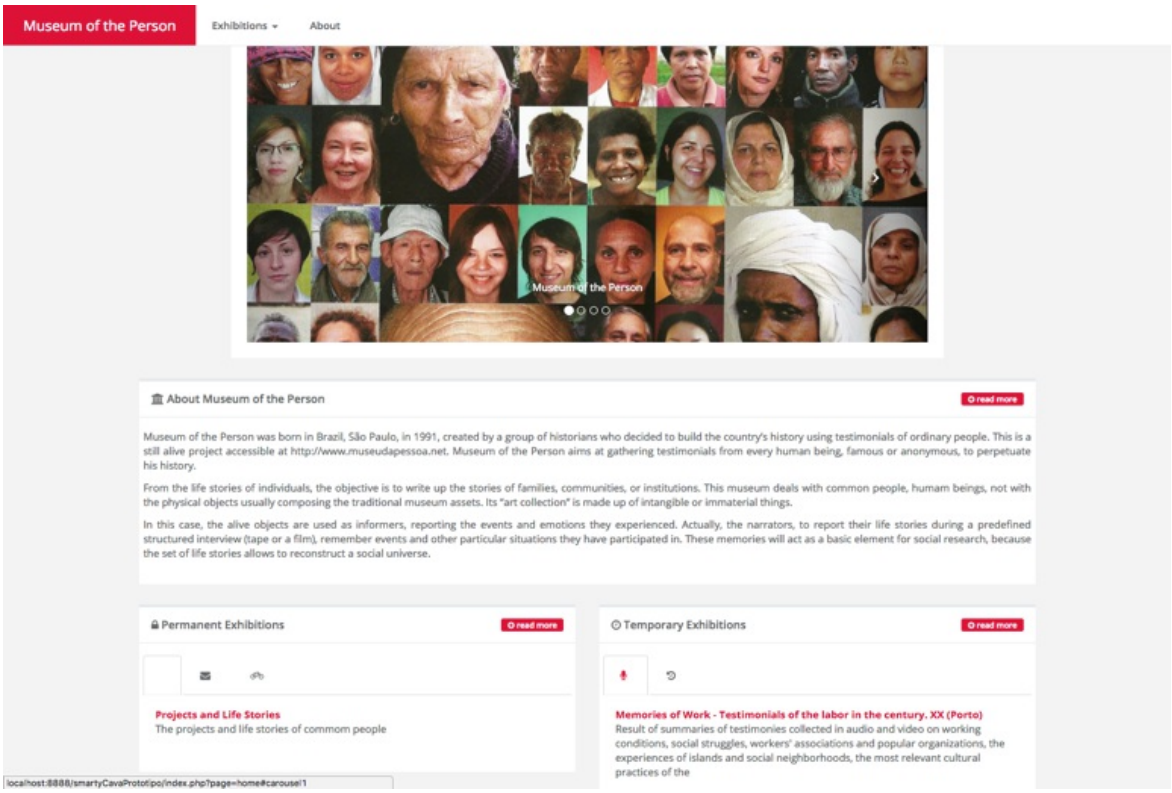

Figura 9. Museu da Pessoa – Espaço Virtual de Aprendizagem gerado pelo CaVa

5. Conclusão

A descrição dos Espaços Virtuais de Aprendizagem em uma linguagem simples, dirigida a um utilizador específico, possibilita a sua geração e capacita o responsável da instituição cultural a focar apenas na exposição do conteúdo (estrutura das salas de exposição) de forma a quepermita ao visitante tirar o máximo proveito da exposição. A abordagem que propomos, baseada na geração automática a partir de uma especificação de alto nível, elimina qualquer necessidade de aprender várias linguagens de programação de uso geral pela pessoa responsável pela instituição cultural; ao aprender a linguagem CaVa^{DSL}, para especificar o Espaço Virtual de Aprendizagem desejado é quanto basta para produzir imediatamente o site sem mais conhecimentos de programação. Tanto a DSL quanto os artefatos de software para lidar com essa linguagem e com a

ontologia nela referida foram introduzidos e explicados ao longo do artigo.

Depois de projetar a DSL e desenvolver os respectivosProcessadores que permitem ao Curador ou ao responsável pela Instituição de Memória descrever e criar as salas de exposição do museu, também incluímos essa linguagem e com a ontologia nela referida arquitetura CaVa proposta, um primeiro módulo (semelhante a um processo ETL) para extrair, transformar e carregar os dados do repositório de origem. O módulo ETL (também chamado de IngestionFunction) também foi descrito ao longo deste artigo.

Muitas ideéias novas surgiram durante o desenvolvimento deste projeto. No entanto, como trabalho futuro, o mais importante é realizar experimentos para avaliar a eficácia e a usabilidade de nossa proposta, bem como aplicar a abordagem a diferentes cenários e a outros casos de estudo.

Bibliografia

Almeida, J. J., J. G. Rocha, P. R. Henriques, S. Moreira, e A. Simões (2001). “Museu da Pessoa – arquitetura”, em *Encontro nacional da associação de bibliotecários, arquivista e documentalistas, ABAD’01*, Universidade do Minho. Disponível em: http://repositorium.sdum.uminho.pt/handle/1822/585. [Consulta: 10 de dezembro de 2018].

Araújo, C. (2016). *Building the Museum of the Person Based on a combined CIDOC-CRM/ FOAF/DBpedia Ontology*. Tese (mestrado), Universidade do Minho.

_____, R. G. Martini, P. R. Henriques, e J. J. Almeida (2016a). “Architectural Approaches to build The Museum of the Person”, em *2016 11th Iberian Conference on Information Systems and*

Technologies (CISTI), pp. 383–388.

_____, C., R. G. Martini, P. R. Henriques, e J. J. Almeida (2016b). “Building the Museum of the Person from RDF Triples and SPARQL”, em *Communications and Innovations Gazette*, v. 1, pp. 1–14.

_____, R. G. Martini, P. R. Henriques, e J. J. Almeida (2018). “Annotated documents and expanded CIDOC-CRM Ontology in the automatic Construction of a Virtual Museum”, em *New Advances in Informations Systems and Technologies*, v. 2.

_____, P. R. Henriques, e R. G. Martini (2017). “Automatizing Ontology Population to drive the navigation on Virtual Learning Spaces em *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 781–786.

Gruber, T. R. (1993).“Toward principles for the design of ontologies used for knowledge sharing”, em *International Journal of Human-Computer Studies*, v. 42, n. 5-6, pp. 907–928.

Martini, R. G., C. Araújo, J. J. Almeida, e P. R. Henriques (2016). “OntoMP, AnOntologyto Build the Museum of the Person”, em *New Advances in Information Systems and Technologies*, v. 2, pp. 653–661.

Parr, T. (2013). *The Definitive ANTLR 4 Reference*. Raleigh: Pragmatic Bookshelf.

Schweibenz, W. (2004). “The development of virtual museums”, em *ICOM*, v. 57, n. 3.

Simões, A. e J. J. Almeida (2003). “Histórias de vida + processamento estrutural = Museu da Pessoa”, em *XATA 2003 — XML: Aplicações e Tecnologias Associadas*, Universidade do Minho, p. 16. Disponível em: http://repositorium.sdum.uminho.pt/handle/1822/629. [Consulta: 10 de dezembro de 2018].